МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Физтех-школа физики и исследований им. Ландау

Машинное обучение. Конспект

Автор:
Яренков Александр Владимирович

Долгопрудный
14 августа 2024 г.

1 Глава 1

1.1 Введение

Определим основные понятия и покажем их на примере из таблицы под этим предложением.

Name	Age	Statistics (mark)	Python (mark)	Eye color	Native language	Target (mark)	Predicted (mark)
John	22	5	4	Brown	English	5	1
Aahna	17	4	5	Brown	Hindi	4	5
Emily	25	5	5	Blue	Chinese	5	2
Michael	27	3	4	Green	French	5	4
Some student	23	3	3	NA	Esperanto	2	3

- Датасет/выборка/набор данных - данные, которые нам доступны

  Вся таблица, кроме последней колонки.

- Объект/наблюдение/datum/data point - элемент выборки

  Обычно(всегда ли?) мы предполагаем их независимыми и одинаково распределёнными (н.о.р.).

  Строка в таблице

- Признак/feature - характеристика объекта.

  Все колонки признаков образуют матрицу признаков/design matrix (X) (её столбцы x_l - вектора, имеющие все признаки объекта).

  Первые 6 колонок в таблице

- Целевая (зависимая) переменная/target (y) - переменная, значение которой нас интересует

  Жёлтый столбец в таблице

- Модель - метод обучения. Формально, действующее на признаковом пространстве отображение ($f(X)$)

- Предсказание/prediction ($\hat{y}$) - результат работы модели

  Зелёная колонка в таблице

- Функция потерь/Loss (L) - критерий качества модели. Её значение мы пытаемся уменьшать, чтобы модель была качестсвеннее.

  Примеры функции потерь:

  Mean Squared Error/дисперсия: $MSE(y, \hat{y}) = \|y - \hat{y}\|_2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

  Mean Absolute Error: $MAE(y, \hat{y}) = \|y - \hat{y}\|_1 = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$

  Здесь $N-$ количество наблюдений

- Параметры - Переменные, значение которых модель выбирает сама

  Например, взвешивание данных

- Гиперпараметры - параметры, которые мы фиксируем до начала работы с данными

  Например, глубина решающего дерева

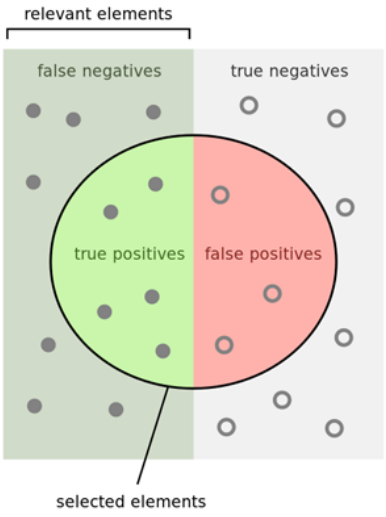
1.1.1 Формальная задача обучения с учителем

Обучение с учителем - обучение с наличием целевой переменной.

Поставим формально задачу обучения с учителем: $\{x_i, y_i\}_{i=1}^N$ −тренировочный датасет(training set), $f(X)$ − модель f, которой на вход подаём матрицу признаков X и получаем предсказание f(X) $= \hat{y}$, $L(X, y, f)$ − Loss function, that should be minimized

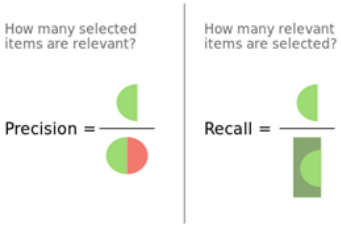
1.1.2 Метрики

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive** TP	**False positive,** FP Type I error
	Predicted condition negative	**False negative,** FN Type II error	**True negative** TN



Precision - доля верно классифицированных объектов (TP) ко всем классифицированным объектам (TP + FP). Она же точность

Recall - доля верно классифицированных объектов (TP) ко всем объектам этого класса (TP + FN). Т.е. мы узнаём сколько значений из класса мы НЕ потеряли.



Есть задачи в которых важнее либо precision, либо recall. Например, заболевшие люди приходят в больницу. Если им назначить лечение, не факт что им станет лучше. Но если им не назначить лечение, то им может стать сильно хуже. Поэтому нам не так важно улучшить состояние каждого (повысить precision - в данном случае доля заболевших людей, у которых улучшилось состояние), как чтобы людям не становилось ещё хуже (т.е. нужно повысить recall - в данном случае доля заболевших людей, у которых не ухудшилось состояние)

Поэтому введём понятие F_β−score. Это метрика, учитывающая важность либо precision, либо recall.

$$F_\beta = (1 + \beta^2)\frac{Precision \cdot Recall}{\beta^2 Precision + Recall}$$

Наиболее частые частные случаи:

$\mathrm{F}_\beta|_{\beta=0} = F_0 = 2 \cdot Precision$
$\mathrm{F}_\beta|_{\beta=+\infty} = F_\infty = Recall$
$\mathrm{F}_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$

1.2 Наивный байесовский классификатор

$\{x_i, y_i\}_{i=1}^N$ - training set. $x_i \in \mathbb{R}^p$ - вектора, $y_i \in \{C_i\}_{i=1}^k$ (Задача классификации на k классов).

А также наивное предположение, которое делает байесовский классификатор наивным: все признаки независимые.

Согласно теореме Байеса,

$$P(y_i = C_j | x_i) = \frac{P(x_i | y_i = C_j) \cdot P(y_i = C_j)}{P(x_i)} \quad (1)$$

Так как все признаки считаем независимыми, то происходит следующая факторизация:

$$P(x_i | y_i = C_j) = \prod_{l=1}^{p} P(x_i^l | y_i = C_j)$$

Таким образом вероятность $P(x_i | y_i = C_j)$ мы поняли как считать. Вероятность $P(y_i = C_j)$ мы сможем посчитать по частоте встречаемости экземпляров класса в тренировочной выборке.

Так как у нас стоит задача классификации, то ответом к ней будет класс, к которому относится объект: $\boxed{C = \arg\max_j P(y_i = C_j | x_i)}$ - аргумент, при котором вероятность максимальна. Заметим, что максимизируем мы по различным классам, а знаменатель выражения (1) не зависит от j, поэтому для задачи максимизации он нам не нужен и мы его просто выбросим из рассмотрения. При желании его посчитать, можно воспользоваться формулой полной вероятности.

1.3 Градиентная оптимизация

1.3.1 Безусловная оптимизация

Пусть модель зависит от параметров w: $f_w(X)$. Наша задача в повышении качества нейронной сети. Математически это качество выражается в виде функции качества, или обратной к ней - функции потерь. Т.о. наша задача состоит в минимизации функции потерь.

Для этого подберём параметры модели, при которых функция потерь будет наименьшей (**это и есть оптимизация**). Так как градиент - это направление наискорейшего роста, то будет подбирать параметры модели следующим образом:

$$w_{n+1} = w_n - \alpha \nabla_w L(y, f_w(X))$$

где (w_n) - последовательность параметров модели (при $n \to \infty$ последовательность должна сходиться к локальному минимуму), w_0 - начальные значения параметров, α - шаг оптимизации/learning rate - некоторый коэффициент, который может меняться на каждом шагу.

1.3.2 Условная оптимизация

Условная оптимизация - это оптимизация параметров, на которые наложены ограничения.

Например, в нашей задаче есть n классов и в каждом из них какое-то количество объектов. Вероятность посмотреть на элемент $i_{\text{того}}$ класса равна p_i. Эти вероятности и будем считать параметрами. На них есть понятное ограничение: $\sum_{i=1}^{n} p_i = 1$. Это простой случай, т.к. это ограничение со знаком равенства, а значит будем применять метод множителей Лагранжа (3й семестр мат. анализа) для нахождения условных экстремумов.

В качестве основы для функции Лагранжа выберем энтропию (*почему?*) $S = -\sum_k p_k \ln p_k$

Тогда функция Лагранжа $\mathcal{L} = -\sum_k p_k \ln p_k + \lambda(1 - \sum_k p_k)$

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial p_i} = -\ln p_i - 1 + \lambda = 0 \\ \\ 1 - \sum\limits_k p_k = 0 \end{cases}$$

$$\begin{cases} \ln p_i = \lambda - 1 => p_i = \exp(\lambda - 1) \\ \\ 1 - \sum\limits_k p_k = 0 => 1 - \sum\limits_k \exp(\lambda - 1) = 0 => 1 - k\exp(\lambda - 1) = 0 => \lambda = 1 - \ln k \end{cases}$$

Таким образом $\ln p_i = -\ln k$ или $p_i = \frac{1}{k}$. Мы нашли кандидатов на условный экстремум и дальше делаем с ними тоже самое, что и для градиентной оптимизации.

1.4 Методы регрессионного анализа

1.4.1 Линейные модели и линейная регрессия

Модель $f(X)$ назвается линейной, если она является линейной комбинацией признаков и какого-то числа.

Поставим формально задачу линейной регрессии: $\{x_i, y_i\}_{i=1}^N -$ training set , $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$

$f(X) = \omega_0 + \sum\limits_{k=1}^{p} \omega_k x_k$

Свободный член w_0 называют **bias term**, т.е. параметр смещения. Потому что в линейной функции свободный член отвечает за смещение прямой относительно начала координат.

Эта запись выглядит довольно громоздко. Её можно упростить с помощью скалярного произведения признаков на веса. Но тогда останется bias term w_0. Поэтому, чтобы писать только скалярное произведение введём следующие обозначения.

Обозначим $\tilde{X} = (1,\, x_1,\, x_2,\, ...\, x_p)^T$, а $W = (w_0,\, w_1,\, w_2,\, ...\, w_p)^T$ - вектор весов. Тогда $f(X) = \tilde{X}^T W$

Для решения задачи нам нужно найти такие веса W, которые минимизируют функцию потерь. Эти искомые веса $\hat{W} = \arg\min\limits_w L$. Тогда решением задачи/предсказанием модели будет $\hat{Y} = XW$

Обычно задачи машинного обучения не имеют аналитического решения, но если мы выберем функцию потерь $L = \|Y - XW\|_2^2$ и будем её минимизировать, то по сути мы будем применять МНК.

$L = (Y - XW)^T(Y - XW)$

$\nabla_w L = \nabla_w(Y^TY + W^TX^TXW - Y^TXW - W^TX^TY) = 0 + X^TXW \cdot 2 - X^TY \cdot 2 = 0 =>$
$W = (X^TX)^{-1}X^TY$. Это решение существует, только есть матрица X^TX обратима, т.е её детерминант не равен 0, т.е все признаки линейно независимы. Есть даже **Теорема Гаусса-Маркова**, которая утверждает что если $Y = XW + \varepsilon$, где $\varepsilon - (\forall\ i\ \mathbb{E}\varepsilon_i = 0)$, дисперсии конечны, а $\forall$ i, j $cov(\varepsilon_i, \varepsilon_j) = 0$, то решение найденное по МНК - оптимальное несмещённое решение.

Но что делать в случае когда признаки зависимые?

1.4.2 Регуляризация

Т.к. признаков стало меньше, то для сохранения единственности решения необходимо ввести дополнительные ограничения. Такое введение ограничений называется регуляризацией.

Например, можно добавить диагональную матрицу для получения линейной независимости и самый простой способ - добавить единичную матрицу. $W = (X^TX + \lambda E)^{-1}X^TY$. На самом деле такой вектор весов это будет решение с помощью МНК для следующей функции потерь: $L = \|Y - XW\|_2^2 + \lambda^2\|W\|_2^2$. Это есть L_2 регуляризация. Частный случай L_p регуляризации.

Вообще L_2 регуляризация пытается найти минимум используя все признаки, т.к. имеет меняющий знак в нуле линейный градиент, а L_1 регуляризация пытается занулить наибольшее количество весов, т.к. имеет меняющий знак в нуле константный градиент.

1.4.3 Задача классификации

Введём понятие правдоподобия. Пусть наша модель работает с параметрами θ. Тогда функция правдодобия - это вероятность $P(X,Y|\theta)$ пронаблюдать такие X и Y при параметрах θ. Логично, что функцию правдоподобия нужно максимизировать. Если X и Y одинаково распределённые $\forall i$ x_i и y_i независимые, то $P(X,Y|\theta) = \prod_i P(x_i,y_i|\theta)$. Далее мы можем воспользоваться монотонной функцией логарифм, т.е. максимизация логарифма величины это та же задача, что и максимизация самой величины. Тогда мы будем рассматривать $\sum_i \ln P(x_i,y_i|\theta) \to \max_\theta$

1.4.4 Логистическая регрессия

Регрессия называется логистической, т.к. есть "0"и "1". Соответсвенно имеем задачу бинарной классификации. Пусть $p \in [0,1]$ - вероятность принадлежности к одному из классов. Но мы пока умеем решать только задачу регрессии, и предсказание является элементом $\mathbb{R}$. Поэтому нам нужно провести какие-то преобразования с вероятностью p, такие что сохранится монотонность вероятности и новое выражение будет принимать значение из $\mathbb{R}$.

Один из способов следующий:

$$p \to \frac{p}{1-p} \in [0,\infty) \to \ln \frac{p}{1-p} \in \mathbb{R}$$

И уже это выражение можно приравнять к предсказанию линейной регрессии $f(X)$ (см. 4.1). Выражая отсюда p, получим $p = \frac{1}{1-e^{-f(X)}}$. Полученная функция называется **сигмоида**. $\sigma(f(X))$

Для задачи бинарной классификации определим понятие *Margin* (сокр. M, по-русски отступ).Margin показывает насколько глубоко в собственном классе находится точка. Введём каждому из двух классов метку: положительную 1 или отрицательную -1. Тогда $M =$ метка класса $\cdot f(X)$.

Итак, вероятность принадлежности к одному из классов, согласно выкладкам выше, есть $\sigma(f(X)) = \sigma(M)$. Тогда вероятность принадлежности к другому классу *с отрицательной меткой* есть $1 - \sigma(f(X)) = \sigma(-f(X)) = \sigma(M)$.

Таким образом, для максимизации функции правдоподобия, нам нужно максимизировать $\sum_i \ln P(x_i,y_i|\theta) = -\sum_i \ln\big(1-\exp(-M_i)\big)$ или $\sum_i \ln\big(1-\exp(-M_i)\big) \to \min_w$. Последнюю называют *логистической функцией потерь*, которую минимизируют.

Таким образом, **логистическая регрессия описывает линейную разделяющую поверхность, но не является линейной моделью**!

1.4.5 Многоклассовая классификация

Чаще всего пользуются следующим методом: с помощью логистической регрессии относят данные к каждому классу по отдельности. Т.е. сколько классов, столько логистических регрессий и нужно будет построить. А потом логично с помощью биссектрис (равноудаление от линий, полученных в результате регрессии) углов добавить оставшиеся данные.

1.5 Метод опорных векторов (SVM)

1.5.1 Линейный

Начнём с простой выборки из двух классов, в которой гиперплоскость разделяет классы с нулевой функцией потерь, где в качестве функции потерь мы выбираем эмпирическую функцию ошибок, т.е. количество неправильно классифицированных точек. Пусть это будут, опять же, положительный и отрицательный класс с соответсвующими метками классов, тогда margin всегда будет положительный.

Отнормируем веса линейной регрессии таким образом, чтобы $\min\limits_i M_i = 1$. Будем здесь считать, что $W \in \mathbb{R}^n$. Это значит, что для любых точек $|W \cdot X - w_0| \geqslant 1$. В этом неравенстве мы по сути берём проекцию на нормаль, а значит в нём заключён margin. Ведь по своему смыслу margin это глубина нахождения точки в классе и мы хотим максимизировать его для получения наиболее хорошей модели. Поэтому найдём проекцию на нормаль ближайшей точки с одной стороны и ближайшей точки с другой стороны. Одна из проекций получится отрицательной, т.к. будет смотреть против нормали. Тогда вычитая одно из другого получим $\frac{(x_+ - x_-)\cdot W}{\|W\|} \geqslant \frac{2}{\|W\|} \to \max$. Видим, что эта оценка, которую мы максимизируем, не зависит от положения точек. А значит изменяя норму W, мы меняем и расстояние между классами.

Теперь формализуем задачу:

$$\begin{cases} \frac{\|W\|^2}{2} \to \min\limits_{W,w_0} \\ M_i \geqslant 1 \end{cases}$$

Теперь обобщим задачу на случай, когда невозможно провести линейную регрессию с нулевым лоссом. В таком случае margin должен уменьшиться (в противном случае это только усиливает рафинированность задачи). Поэтому добавим в формулировку $\xi_i \geqslant 0$. $M_i \geqslant 1 \ \to M_i \geqslant 1 - \xi_i$. Так как мы хотим совершать наименьшее число ошибок, т.е. минимизировать лосс, то и туда нужно добавить сумму всех этих ξ_i. Итого получится следующая формулировка задачи:

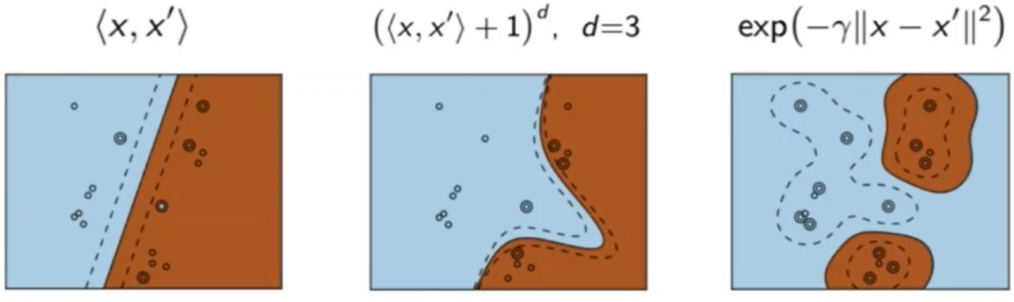
$$\begin{cases} \frac{\|W\|^2}{2} + C\sum\limits_{i=1}^{l} \xi_i \to \min\limits_{W,w_0,\xi} \\ \forall i \ M_i \geqslant 1 - \xi_i \\ \forall i \ \xi_i \geqslant 0 \end{cases}$$

Выражая отсюда ξ_i и убирая его получим следующую более простую формулировку задачи:

$$\frac{\|W\|^2}{2} + C\sum_{i=1}^{l}(1 - M_i) \to \min_{W,w_0}$$

Таким образом, получим так называемый *Hinge loss*: $L = \sum\limits_{i=1}^{l}[M_i < 0] \leqslant \sum\limits_{i=1}^{l}(1 - M_i) + \frac{1}{2C}\|W\|^2$

1.6 Нелинейный



Если мы поменяем скалярное произведение, то изменим метрическое пространство (матрица Грама). И на самом деле мы будем пользоваться тем же самым линейным SVM и будем получать гиперплоскости, но уже в другом пространстве, признаковом пространстве.

1.7 Оценка качества классификации

1.7.1 Коэффициент детерминации

$$SSE = \sum_i (y_i - \hat{y}_i)^2$$

$SST = \sum_i (y_i - < y >)^2$ Тогда определим коэффициент детерминации как $R^2 = 1 - \frac{SSE}{SST}$. Из формулы видно, что чем лучше наша модель, тем ближе коэффициент детерминации к 1. Однако, чем больше данных у нас будет, тем меньше будет каждое слагаемое в SST, а значит R^2 тоже будет увеличиваться. Таким образом, сравнивать качества моделей с разным количеством данных с помощью коэффициента детерминации некорректно.

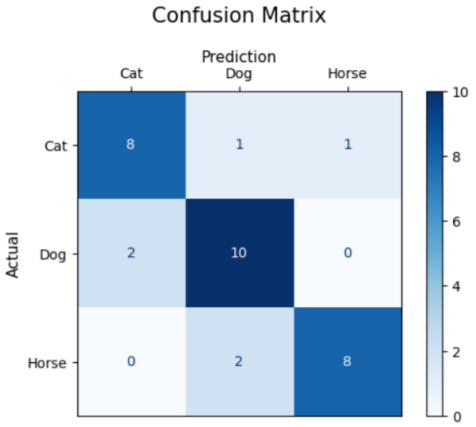
1.7.2 ROC-кривая

Модели машинного обучения могут создаваться для разных целей. Например, для определения болен человек или здоров. И эта модель может выдавать какое-то число (как в случае регрессии). Для примера, это может быть число от 1 до 10, где выше 7.5 - болен. Это число 7.5, по которому меняется отнесение объекта к классу называется *порогом*. И при разных порогах, очевидно, мы будем получать разные Presicion и Recall (взять банально классификатор, относящий все объекты только к одному классу). Так вот меняя этот самый порог, мы будем получать различные точки на Precision-Recall диаграмме. В нашем примере в случае порога равного 0, всегда предсказывается что человек болен. А в случае порога равного 10, всегда предсказывается что человек здоров. В случае чистой линейной регрессии без дальнейших преобразований это пороги $+\infty$ и $-\infty$.

Таким образом, мы получаем ступенчатую ROC-кривую в координатах Precision-Recall, всегда выходящую из 0, и заканчивающуюся в точке (1, 1). При увеличении количества точек эта кривая будет лишь сглаживаться, т.е. AUC (area under curve) должен остаться примерно тем же. Поэтому для оценки качества классификатора используют значение ROC-AUC. Чем оно выше, чем лучше классификатор. Если ROC-AUC = 0.5, т.е. количество отнесений объектов к классам не меняется, то логично что это классификатор, который совершенно случайно даёт предсказания. Т.е бесполезный :)

1.7.3 Матрица ошибок (Confusing matrix)

Матрица ошибок - это квадратная матрица размера $n \times n$, где n - количество классов. Эта матрица показывает количество правильно и неправильно классифицированный объектов. Вот пример, как может выглядеть матрица ошибок.



1.8 Cross-validation

Обычно выборку данных делят на 3 подвыборки: train, valid, test. На train выборке модель обучается, на valid выборке проверяется качество обученной модели Обычно valid выборка является составляющей train выборки. Затем на test выборке, которую модель никогда до этого не видела (в отличие от valid) окончательно проверяется качество обученной модели.

Кросс-валидация - это когда мы берём разные в train подвыборки (valid) несколько раз, тем самым обучаясь на train − valid и валидируясь на valid каждый раз на разных поднаборах данных. Например, можно поделить выборку на 5 частей и обучить модель 5 раз, валидируясь на каждой из 5 частей по очереди и, соотвественно, обучаться на оставшихся 4 частях.

Есть и более сложные методы кросс-валидации. Например, можно каждый раз брать по какому-то "кусочку"из 5 частей из примера выше так, чтобы в сумме на валидации была $\frac{1}{5}$ train набора данных. Это называется стратифицированной выборкой. А можно вообще случайным образом выбирать valid подвыборку из train.

Однако так не получится делать, если данные на которых предстоит обучаться машине это временные ряды. Обучаться на будущем и предсказывать прошлое не имеет смысла.

1.9 Решающие деревья

Решающее дерево - это набор условий, по которым разделяются данные в разные классы. Всё начинается с корня дерева - начального условия. Затем по веткам переходим либо в новые условия и ветвимся дальше, либо попадаем в конец дерева (лист), где уже определяется класс.

Осталось понять как выбирать эти условия. Пусть мы дошли до ветки с выборкой Q, которая ветвится на выборки L и R. Пусть H - критерий по которому мы будем выбирать условие. Этот критерий выбирается так, что чем меньше его значение, тем предпочительнее будет выбрать такое условие. Тогда нам нужно будет просчитать этот критерий для каждого возможного условия в подвыборках L и R и, так как эти выборки могут быть разного размера, это необходимо учитывать. По итогу, мы должны минимизировать следующее выражение на каждом ветвлении дерева:
$\frac{|L|}{|Q|}H(L) + \frac{|R|}{|Q|}H(R)$.

Какие же бывают критерии? Для задачи регрессии можно взять, например, MSE. Для задачи классификации 2 наиболее известных это критерий информативности и критерий Джини. С первым всё понятно, мы используем статистическую энтропию, у которой минимум в вырожденном распределении и максимум в равномерном. Первое легко показать, а второе было доказано ранее в 3.2. Энтропию, как меру хаоса, мы стараемся свести к минимуму.

Теперь о критерии Джини. При выборе с возвращением вероятность взять одно и то же данное 2 раза обозначим p_k^2. Вероятность не вытащить 2 данных из одного класса соответственно $1 - p_k^2$. Пусть у нас N классов. Тогда вероятность того, что мы не вытащим 2 подряд данных из одного класса $G = 1 - \sum_{k=1}^{N} p_k^2$. Это, как и энтропия, ещё один способ измерить разнородность данных.

Решающие деревья можно усекать (англ. pruning). Можно это делать до построения дерева, например, ограничить максимальную глубину дерева или размер листа (количество элементов в нём). Либо уже усечь уже обученное дерево с минимальными потерями качества модели. Это нужно для того, что решающие деревья очень склонны к переобучению и их нужно усекать.

Решающие деревья обладают ещё одним полезным свойством, а именно что на пропуски в данных можно забить. Если про какой-то признак у некоторых данных у нас нет информации, то мы можем эти данные отправить в обе ветки, посчитать взвешенное среднее предсказаний и тогда итоговое предсказание будет следующим: $\hat{y} = \frac{|L|}{|Q|}\hat{y}_L + \frac{|R|}{|Q|}\hat{y}_R$.

Предсказание решающего дерева можно записать в следующем виде: $\hat{y}(x) = \sum_{i=1}^{N} w_i I[x \in C_i]$ - взвешенная сумма индикаторов принадлежности элемента x к каждому из классов. Этим действием мы сформировали новое признаковое описание данных, где в качестве признаков выступает

принадлежность к конкретному классу. Тогда написанная выше формула всё равно что линейная модель.

1.10 Техники ансамблирования

1.10.1 Bootstrap и Bagging

Bootstrap - это метод создания новых датасетов на основе имеющегося. Как он работает? Из исходного датасета берётся с возвращением N элементов. Получаем новый бустрапированный датасет. Если мы повторим так M раз, то получим M новых датасетов и можем на каждом из них обучить разные модели.

Усредним предсказания всех моделей и получим $< \hat{y} >= \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i$. Дисперсия $i_{\text{той}}$ модели по определению $\sigma_i^2 = \mathbb{E}(\hat{y}_i - y)^2$. А усреднённая дисперсия предсказания моделей $\sigma^2 = \frac{1}{M}\sum_{i=1}^{M}\sigma_i^2$. Сделаем предположение, что все ошибки σ_i несмещённые (мат ожидание всех ошибок равно нулю) и нескоррелированные. И обозначим $\varepsilon_i = \hat{y}_i - y$. Тогда дисперсия усреднённого предсказания моделей

$$\mathbb{E}\left(\frac{1}{M}\sum_{i=1}^{M}\varepsilon_i\right)^2 = \frac{1}{M^2}\mathbb{E}\left(\sum_{i=1}^{M}\varepsilon_i^2 + \sum_{i\neq j}\varepsilon_i\varepsilon_j\right) = \frac{1}{M^2}\mathbb{E}\left(\sum_{i=1}^{M}\varepsilon_i^2\right) = \frac{1}{M}\cdot\frac{1}{M}\left(\sum_{i=1}^{M}\mathbb{E}(\hat{y}_i - y)^2\right) = \frac{1}{M}\sigma_i^2$$

То есть мы получили, что дисперсия падает в M раз, если взяли M моделей.

Тесно связанное с этим понятие *Bagging* = **B**ootstrap **agg**regat**ing**. Это и есть метод обучения, когда берётся M моделей и обучается на M бутстрапированных выборках. Этот метод очень хорошо подходит для решающих деревьев, т.к. они склонны к переобучению и получаются все совершенно разные, обученные на бутстрапированных выборках.

Но сколько моделей нам вообще нужно обучить? При увеличении количества моделей в какой-то момент они будут обучаться на довольно близких датасетах и станут похожими друг на друга. В таком случае ошибки станут скоррелированными, т.е. линейно зависимыми и функция ошибки перестанет падать.

1.10.2 RSM и Случайный лес

RSM = Random Subspace Method. Этот метод ансамблирования заключается в том, что каждый раз мы делаем выборку не данных из датасета, как в bootstrap, а делаем выборку из признакового пространства. И каждый раз наша модель будет выбирать, например, лучший не из 8 возможных признаков, а из 5.

В случае решающий деревьев, RSM позволяет сделать модели ещё более непохожими друг на друга, что усиливает независимость моделей и соответсвенно их ошибок. *Случайный лес* - это одновременное применение bagging и RSM.

1.10.3 Blending

Blending - это техника, когда мы обучаем несколько моделей на одной тренировочной выборке, а затем из предсказаний этих моделей формируем новое признаковое пространство для данных из валидационной выборки. Это признаковое пространство будет размерности такой же, какое число обученных разлчиных моделей. Затем мы обучаем новую выпуклую модель, являющуюся взвешенным средним всех предыдущих моделей, то есть сумма всех весов равна 1 и значение каждого веса от 0 до 1.

Имеет смысл делать blending только с уже неплохо работающими моделями, иначе просто нет смысла усреднять посредстсвенные результаты. Однако линейной комбинации моделей не всегда достаточно.

1.10.4 Stacking

Помимо недостаточности линейной комбинации, ещё не хочется и делить данные на train и valid выборки. С этим всем может помочь cross validation. Мы делим датасет на фолды и обучаем все модели на разных фолдах и делаем предсказания для всего датасета. Теперь можно использовать наши модели на тестовой выборке. В итоге у нас получается новое признаковое пространство для каждого экземпляра целевой переменной из датасета, состоящее из предсказаний всех моделей. Весь этот процесс и называется *stacking*.

Далее можно делать stacking над stacking'ом. Обычно, глубина вложенности составляет 3-4 итерации stacking'а.

Какие есть проблемы? Из-за того, что мы обучаем модели на разных данных, мы получаем по-разному работающие модели. И предсказания для всего датасета, сложенное из предсказаний обученных по-разному моделей, будут уже по факту НЕ одинаково распределённые. Т.е. наше новое признаковое пространство состоит из разно распределённых величин. Чтобы эти распределения не были сильно разными, применяют регуляризацию. Также тяжело объяснить как модель, обученная над несколькими вложенными stacking'ами, делает предсказания.

1.11 Оценка значимости признаков

Бывает нам нужно понять, а насколько вообще важны признаки для модели. Самый очевидный способ - это перемешать столбец с этим признаком и посмотреть насколько ухудшится предсказание модели. Но с большими данными это будет происходить очень долго. Рассмотрим сначала интуитивно очевидные методы оценки значимости признаков для деревьев.

Gain - в каждой ветке дерева решений выбирается порог для максимизации критерия информативности. Поэтому мы просто будем считать для каждого признака на сколько он увеличил информативность во всех ветках дерева.

Cover - этот метод заключается в том, что чем чаще происходит ветвление и чем больше данных ветвятся по этому признаку, тем выше значимость этого признака

Shap - ...

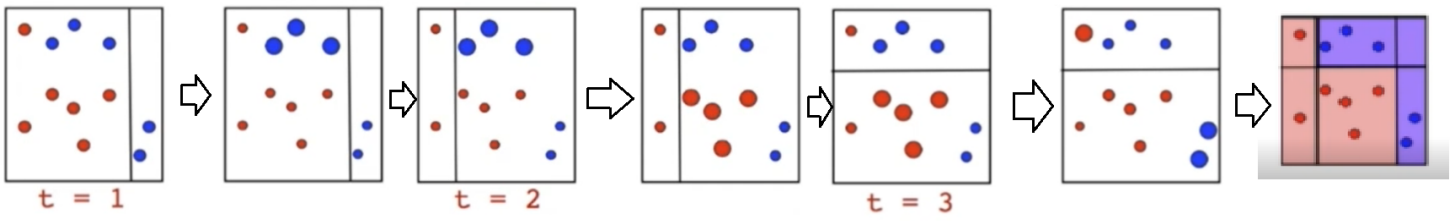
Для линейных моделей оценить значимость признаков можно по весовым коэффициентам. Чем больше по модулю коэффициент, тем сильнее в его направлении наклон прямой, т.е. тем значимее признак. При этом, конечно же, нельзя забывать о нормировке данных. Иначе, если один признак принимает значения от 0 до 1, а другой от 100 до 1000, то и коэффициенты могут получиться столь же разного порядка, хотя значимость у них на самом деле будет одинаковая.

1.12 Бустинг

Ранее мы рассматривали техники ансамблирования и обучали модели параллельно друг с другом и усредняли их результаты. Но можно ведь и обучать модели последовательно, друг за другом. При этом каждая следующая модель будет учитывать ошибки предыдущей. Это и есть **бустинг**. Как не трудно догадаться, такое мы можем проворачивать и с самыми простыми моделями, постепенно увеличивая их качество.

1.12.1 AdaBoost

Рассмотрим простой пример. Пусть у нас будет решающий пень (решающее дерево глубины 1). С его помощью мы можем строить только линейные разделяющие поверхности, т.к. идём либо "налево"по ветке, либо "направо"по ветке. Далее неправильно классифицированным данным мы меняем веса на бо́льшие, а у правильно классифицированных точек на меньшие. Далее мы берём все наши модели с нужными весами и получаем нелинейную разделяющую поверхность только с помощью одной модели, способной строить лишь линейные поверхности.



На рисунке t - это номер модели, а самая правая картинка - взвешенное среднее трёх моделей, построенных с помощью бустинга.

Для такой модели в качестве функции потерь возьмём экспоненциальную. Т.е. $L = e^{-M}$, где margin $M(x) = y \cdot f(x)$, где y - это метка класса. Тогда на шаге t функция потерь для j-той точки $L = e^{y_j \sum_{i=1}^{t-1} \omega_i f_i(x_j)} \cdot e^{y_j \omega_t f_t(x_j)} = \Omega \cdot e^{y_j \omega_t f_t(x_j)}$. Здесь выделенная Ω - уже константа, т.к. все предыдущие модели в последовательности мы обучили, а нынешнюю обучаем сейчас. И мы хотим минизировать на каждом шаге, в т.ч. на этом, функцию потерь. Выходит Ω выступает в роли функции потерь от предудыщих $t-1$ моделей. Причём неправильно классифицированные точки имеют большой вес, а правильно классифицированные точки имеют малый вес.

Экспонента, это не всегда хорошо. Например, если у нас есть выбросы, то экспонента будет сильно большой и будет очень плохо и не правильно влиять на построение хорошей модели

1.12.2 Градиентный бустинг

Будем ещё одним способом строить модель на t-том шаге. Причём модели мы не берём фиг пойми откуда, а сужаем как-то круг, в котором ищем модели. То есть они у нас будут параметризованы параметрами . Т.е. у нас уже есть $t-1$ моделей со своими предсказаниями. Далее мы, имея в целях уменьшить функцию потерь, посчитаем её частную производную по предсказаниям моделей во всех точках и возьмём со знаком минус, чтобы получить антиградиент и двигаться в сторону наискорейшего убывания лосса. И поэтому это значение антиградиента можно использовать в качестве предсказания новой модели. А также использовать для нахождения оптимальных параметров для построения модели на t-том шаге.

$$r_{jt} = -\frac{\partial L}{\partial f_t(x_j)}$$

$$\hat{\theta}_t = \arg\min_{\theta} \sum_{j=1}^{n} (r_{jt} - f(x_j, \theta))^2$$

Вес же модели $f(x, \hat{\theta}_t)$ найдём из задачи минимизации функции потерь.
В качестве базы индукции можем выбрать любую, хоть константную, функцию.

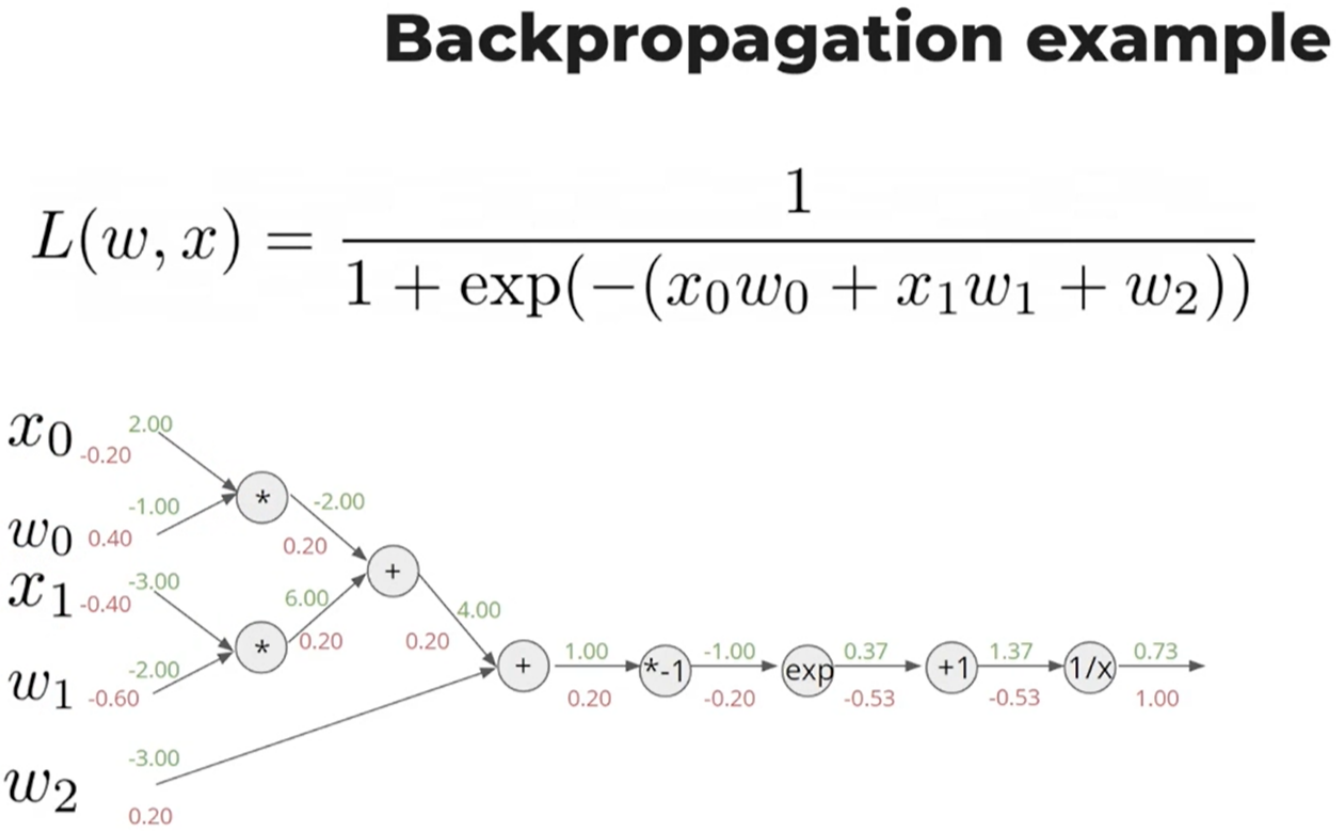
Градиентный бустинг работает гораздо медленнее ранее рассмотренных техник, т.к. раньше мы обучали модели параллельно, а теперь - последовательно. Т.е. новая модель начнёт обучаться только тогда, когда закончит обучаться старая. Параллельное обучение запустить не получится. Также градиентный бустинг склонен к переобучению, т.к. мы буквально пытаемся уменьшить функцию ошибки для данного датасета. Но при этом всём, при грамотном его использовании, градиентный бустинг позволяет хорошо повысить качество обучаемой модели.

2 Глава 2. Нейронные сети

Слой - отображение из одного пространства в другое. Делает преобразование с данными. Нейронная сеть - это последовательность слоёв, среди которых выделяют входной слой (сами данные) и выходной слой (предсказание сети). Нейронная сеть обычно дифференцируемая.

2.1 Механизм обратного распространения ошибки

Рассмотрим на простом примере что это такое. Пусть наша нейронная сеть это последовательность и линейной модели и логистической регрессии. Тогда поочерёдно расчитаем все шаги (на рисунке зелёным цветом. Граф и формула биективны). Затем, идя с конца, стартуем с 1 и считаем производные на каждом шаге и подставляем в них полученные при прямом проходе значения (на рисунке красным цветом).



2.2 Функции активации

Функция активации - функция, которая преобразует данные нелинейным образом. Функции активации должны быть дифференцируемыми.

2.2.1 Сигмоида

$\frac{1}{1+exp(-x)}$. Плюсы:

- Переводит $\mathbb{R} \to [0, 1]$ и поэтому может предсказывать вероятность.

Минусы:

- Децентрирует данные (если среднее было 0, то станет не 0)
- Обнуляются градиенты на концах

- Имеет в себе экспоненту и поэтому может плохо/долго считаться

Из выше написанного следует, что сигмоида хороша разве что для задачи промежуточной бинарной классификации. Например, когда нужно решить какие данные важные и их пустить дальше в рассмотрение, а какие не важные и их стоит выкинуть.

2.2.2 Гиперболический тангенс

$\tanh x$. Плюсы:

- Переводит $\mathbb{R} \to$ [-1, 1]. Данные становятся ограниченными.

Минусы:

- Обнуляются градиенты на концах
- Имеет в себе экспоненту и поэтому может плохо/долго считаться

Похожа на сигмоиду, но оставляет данные центрированными. Поэтому используется, когда данные необходимо подавать со значениями в определённом диапазоне.

2.2.3 ReLU

$\max(0, x)$. Плюсы:

- Очень легко считать производную. Достаточно просто сравнить значение с 0.
- Не обнуляются градиенты при $x > 0$.

Минусы:

- Децентрирует данные
- Обнуляются градиенты при $x < 0$.

Из-за быстроты работы хорошо использовать как baseline.
У этой функции есть улучшения. Например Leaky ReLU или ReLU с утечкой. $max(qx, x), 0 < q < 1$. В этом случае градиенты ещё и при $x < 0$ не обнуляется, что хорошо. Есть также попытка сгладить эту функцию и называется она ELU. Но сглаживание происходит с помощью экспоненты, что не есть хорошо.

2.3 SGD и его доработки. Optimizer

Когда данных становится достаточно много, то классический градиентный спуск считается довольно долго. И хотелось бы ускорить обучение модели. Для этого применяется следующий метод. **SGD - Stochastic Gradient Descent**, он же стохастический градиентный спуск. Это когда мы считаем градиент для случайной подвыборки (batch), а не для всей выборки как в градиентном спуске. Ну а затем применяем классический шаг градиентного спуска. Таким образом мы получаем оценку градиента всей выборки. Такой способ позволяет ускорить работу модели за счёт уменьшения количества данных, на которых считается градиент.

Соответственно хочется, чтобы размер batch'а был как можно меньше для наиболее быстрого обучения модели. Но тогда, при уменьшении размера batch'а, в какой-то момент данные могут начать приходить только из разных частей распределения и batch'и будут сильно зашумлённые. То есть градиент может начать бегать случайным образом и никуда не сойтись. Но есть некоторые техники, которые позволяют с этим справляться.

SGD и все его доработки называются одним словом - *optimizer*. И это название используется в основных библиотеках для глубокого обучения, например, torch или tensorflow.

2.3.1 Momentum

Первая идея заключается в добавлении "инерции"градиенту. Невозможно быстро изменить направление на 90 градусов при достаточно большом импульсе в нынешнем направлении.

$$\begin{cases} v_{t+1} = \rho v_t + \nabla f(x_t) \\ x_{t+1} = x_t - lr \cdot v_{t+1} \end{cases}$$

Здесь, вместо градиента мы теперь вычитаем импульс, а сам импульс это теперь не просто градиент, но и информация о прошлом, о том в какую сторону в целом был градиент. ρ здесь - это число от 0 до 1 не включительно. Его наличие позволяет постепенно "забывать"что было раньше. Т.е. вклад от импульса на начальных шагах будет идти с коэффициентом ρ^t, что будет близко к 0.

Добавление инерции градиенту хорошо тем, что позволяет быстрее проходить ровные участки, когда функция (например, функция потерь, которую часто мы минизируем) практически не изменяется (они же плато). Однако тогда в какой-то момент мы можем накопить достаточно большой импульс, много больший чем сам градиент в этой точке. Тогда считать градиент в этой точке не будет иметь смысла. В таком случае посчитаем градиент в следующей точке и получим **Nesterov momentum**:

$$\begin{cases} v_{t+1} = \rho v_t + \nabla f(x_t + \rho v_t) \\ x_{t+1} = x_t + v_{t+1} \end{cases}$$

2.3.2 Нормировка градиентов

Здесь главная идея в том, что если градиент маленький, то можно сделать шаг в этом направлении побольше. И наоборот, если градиент большой, то чтобы далеко не уйти и не проскочить минимумы, будем делать шаги поменьше. В частности, поэтому такой метод будет очень хорошо работать в седловых точках.

Adagrad:

$$\begin{cases} cache_{t+1} = cache_t + (\nabla f(x_t))^2 \\ x_{t+1} = x_t - lr \cdot \frac{\nabla f(x_t)}{cache_{t+1} + \varepsilon} \end{cases}$$

Здесь ε нужен для того, чтобы мы случайно не поделили на 0. Также в этом простом случае градиент сделан неубывающим. Добавим коэффициент β, который будет характеризовать скорость забывания предыдущего кэша.

RMSProp:

$$\begin{cases} cache_{t+1} = \beta \cdot cache_t + (1 - \beta)(\nabla f(x_t))^2 \\ x_{t+1} = x_t - lr \cdot \frac{\nabla f(x_t)}{cache_{t+1} + \varepsilon} \end{cases}$$

Как и всегда ранее, эти наши доработки можно объединить и получить подобие **Adam**'а.

2.3.3 Momentum & Нормировка градиентов

$$\begin{cases} v_{t+1} = \rho v_t + (1 - \rho)\nabla f(x_t) \\ cache_{t+1} = \beta \cdot cache_t + (1 - \beta)(\nabla f(x_t))^2 \\ x_{t+1} = x_t - lr \cdot \frac{v_{t+1}}{cache_{t+1} + \varepsilon} \end{cases}$$

2.4 Регуляризация в DL

2.4.1 Batch normalization

Batch normalization - это слой стандартизации данных. На каждый новый слой в нейросети приходят данные с распределением, отличным от распределения на предыдущем слое. И при подсчёте,

в т.ч. градиентов, это является большой проблемой. Ведь с самого начала мы эксплуатируем гипотезу независимых и одинаково распределённых наблюдений. Если мы будем на каждом шаге (или через шаг, т.к. не очень сильно меняется распределение за 1 шаг) применять batch normalization слой, то на каждом слое у нас гарантировано будут одинаково распределённые данные, со средним 0 и дисперсией 1. Более того, lr можно ставить теперь больше, не боясь сильного изменения распределения и, поэтому, обучение модели пойдёт гораздо быстрее

2.4.2 Dropout

В нейронных сетях особенно остро стоит проблема переобучения, т.к. обычно есть очень много признаков и переобучиться под них всегда довольно легко. Можно просто добавлять регуляризационный член в Loss функцию, как мы это делали в простых моделях машинного обучения. Но для нейронных сетей доступен свой интересный способ.

На каждом слое мы можем занулить коэффициенты на некоторых нейронах с помощью специального слоя с названием *Dropout*. Ему в параметры передаётся процент нейронов, значения на которых необходимо занулить. И выбираются эти нейроны случайным образом. Из-за того, что переобучение модели может происходить именно на некоторых слоях, dropout хорошо помогает бороться с переобучением. Однако, когда после обучения модели мы подадим ей данные на вход для предсказания, будут задействованы все нейроны. И если, например, мы везде ставили dropout со значением в 50 %, то обобщающая способность модели будет в 2 раза ниже. Т.к. у модели всегда на самом деле при обучении было в 2 раза меньше нейронов. Это как с деревьями, чем больше их глубина, тем более сложные зависимости они могут описывать, т.е. их обобщающая способность будет выше.

2.4.3 Augmentation

Аугментация - это создание модели более устойчивой к некоторым преобразованиям. Например, если мы отразим фотографию котика относительно вертикальной оси, или изменим яркость фотографии, или добавить мелко-дисперсный шум, или повернём картинку на небольшой угол (будто котик сфоткан под углом), то мы всё равно распознаем котика. Поэтому, помимо создания более устойчивой модели к некоторым преобразованиям, можно увеличивать обучающий датасет в несколько раз.